NOVA SCHOOL OF SCIENCE & TECHNOLOGY
DEPARTAMENTO DE INFORMÁTICA

Computer Architecture 2024/25

TPC 2

Delivery: 6pm on April 15, 2025

This homework consists of a programming exercise **to be carried out in a group of no more than two students**. You can clarify general doubts with colleagues but the solution and writing of the code should be strictly carried out by the members of the group. Plagiarism will be punished in accordance with the regulations in force.

Delivery will be through Mooshak using the identifiers of TPC1; it is possible to use new identifiers if the group changes, More details will be contained in a forthcoming CLIP message.

Exercise

In this exercise you must complete the code of a simulator (written in the C language) of an architecture composed of a very simple CPU and a memory. This includes, already implemented, a console where it is possible for a user to give commands to read and write to the central memory, including reading to memory the values represented in a text file. You can also have the instructions in the memory executed. The implemented commands are as follows:

poke EEE X – writes the value X to the *EEE* address.
peek *EEE* – reads the value at the *EEE* memory address. The value is displayed in hexadecimal.

dump *X* – shows the contents of the accumulator and *X* words from memory starting from address 0.

load *FILE* – reads the file named *FILE,* interpreting each line of text as a value that is placed sequentially in program memory.

run – executes the instructions from address 0 of memory.

The *X* and *EEA* values can be given in base 10 or in base 16 if starting with 0x.

The code to complete is in the dorun.c file and you should implement the *fetch*, *decode,* and *execute* loop to simulate the execution of the instructions in memory. Execution should always run from memory address 0 until the HALT statement is executed (see next section).

Machine instructions of the CPU:

The CP uses 16-bit words and has a general-purpose register which is the accumulator. The main memory is organized in 16-bit words; there are 4096 (2^12) words of memory, and each address has 12 bits,

The instructions are fixed in size of 16 bits distributed as follows.

15 – 12	11 – 0
Código da Operação (4 bits)	Endereço (12 bits)

Bits 15-12: *Instruction code*: specifies the instruction to be executed and how the remaining bits should be interpreted.

Bits 11-0: *Constant with 8 bits*: can be an address (unsigned integer) or a value (signed integer)

The supported instructions and their machine codes are shown in a table below. The following conventions are used

Symbol	Meaning
eeeeeeeeeeee	Address with 12 bits; it is an unsigned integer
xxxxxxxxxxxx	Bits ignored
vvvvvvvvvvvv	12 bits that specify a signed integer codifeus in 2s-complement; the value represented is between 2^{11} -1 and -2^{11}

Instruction	**Mnemonic**	**Description**
0000xxxxxxxxxx	HALT	Ends the programa
0001eeeeeeeeee	LOAD address	Accumulator receives the contents of the memory position with address *eeeeeeeeee*
0010eeeeeeeeee	STORE address	The value contained in the accumulador is written in the memory position with address *eeeeeeeeee*
0011eeeeeeeeee	ADD address	The contents of memory position with address *eeeeeeeeee* is added to the contents of the accumulator and the result is stored in the accumulator
0100eeeeeeeeee	SUB address	The contents of memory position with address *eeeeeeeeee* is subtracted from the accumulator and the result is stored in the accumulator
0101eeeeeeeeee	MUL address	The contents of memory position with address *eeeeeeeeee* is multiplied bythe accumulator and the result is stored in the accumulator
0110eeeeeeeeee	DIV address	The contents of the accumulator is divided by the contents of memory position *eeeeeeeeee* is added and the result is stored in the accumulator
0111eeeeeeeeee	JUMP address	PC receives the address eeeeeeeeeee
1000eeeeeeeeee	JZ address	PC receives the address eeeeeeeeeee, only if the accumulator contains 0.
1001eeeeeeeeee	JN address	PC receives the address eeeeeeeeeee, only if the accumulafor contains a negative value, ie, the most significative bit is 1.
1010eeeeeeeeee	CALL address	Current value of PC plus 1 iis stored in the memory position with address *eeeeeeeeee* ; PC receives the value *eeeeeeeeee* plus 1
1011eeeeeeeeee	RETURN address	PC receives the value stored in address *eeeeeeeeeee.*
1100vvvvvvvvvvv	LDI value	Accumulator receives value *vvvvvvvvvvv*. As the accumulator has 16 bits e vvvvvvvvvvv only has 12, a sign extension is performed
1101	Unused code	
1110	Unused code	
1111xxxxxxxxxx	NOP	Does nothing

Example of a program that calculates y = 2^x through an iterative approach. Result *y* is stored in position 0x11; x is the value in position 0x10, and in position 0x12 is the value 1.

```
address code     mnemonic.         comment
0x000:  0x1012   LOAD   0x012    // AC ← 1
0x001:  0x2011   STORE  0x011     // y ← 1
0x002:  0x1010   LOAD   0x010    // AC ← x
0x003:  0x800F   JZ     0x00F    // ends the program, jumping to 0x00F
0x004:  0x4012   SUB    0x012    // AC ← AC - 1
0x005:  0x2010   STO    0x010    // x ← AC
0x006:  0x1011   LOAD   0x011    // AC ← y
0x007:  0x3011   ADD    0x011    // AC ← y + y
0x008:  0x2011   STO    0x011    // y ← 2 * y
0x009:  0x7002   JMP    0x002    // starts a new cycle
0x00A:  0xF000   NOP
0x00B:  0xF000   NOP
0x00C:  0xF000   NOP
0x00D:  0xF000   NOP
0x00E:  0xF000   NOP
0x00F:  0x0000   HALT
0x010:  0    (x  filled with poke)
0x011:  0    (y to be read with peek)
0x012:  1    (to decrement)
```

The supplied *p.code* file contains the code and data of this program in hexadecimal notation, occupying one word of memory per line. Running the simulator, we can load the program by doing load *p.code* and running it. We can check the result by seeing what is in the address variable 0x02 with *peek*. We can change the value of x with the poke command. Example of a 2^5 calculation session:

```
cmd> load p.code
cmd> poke 0x10 5
cmd> run
HALT instruction executed
cmd> peek 0x11
0x11: 0x20
```

Delivery

Delivery must be made on time and the Mooshak system will be used; you should only submit *dorun.c file.*

The program will be compiled with the following command

```
gcc -Wall -std=c11 -o sim sim.c dorun.c
```